Licenciatura Engenharia Informática e Multimédia

Computação Física – CF

Semestre de Verão 2021 / 2022

Relatório Trabalho Prático 3

Docente Carlos Carvalho

Trabalho realizado por:

Fábio Dias, nº 42921

Índice

Índice de Figuras

1. Objetivo

Para este trabalho prático foi-nos pedido a simulação de uma estação meteorológica com informação da temperatura, pressão atmosférica e altitude. Para obter estas informações, recorremos ao sensor BMP180. Para as apresentar, é usado um *display* de duas linhas de dezasseis caracteres.

É nos também pedido para criarmos um *outdoor* que apresente a mesma informação.

Ambos estas exibições de informação devem ser atualizadas após 10 segundos.

2. Desenvolvimento

Dado que usaremos a placa IMU 10 DOF e o display LCD, que ambos usam um protocolo de comunicação I^2C, podemos esquematizar o diagrama de ligações[1][2].

Para os pinos de Serial Data, SDA, estes vão ser ligados ao pino A4 do arduino[1][2]. Por sua vez, os pinos de Serial Clock, SCL, são ligados ao pino A5[1][2]. Ambos os Ground's, GND, são ligados a um dos pinos GND do Arduino. A única diferença entre estas ligações é que a placa IMU 10 DOF é alimentada a 3.3 Volts enquanto o display LCD é alimentado a 5 Volts. (Ver Figura 1 – Diagrama de Ligações).

Embora usemos a placa IMU 10 DOF, vamos apenas usar o sensor integrado BMP180[1][2].

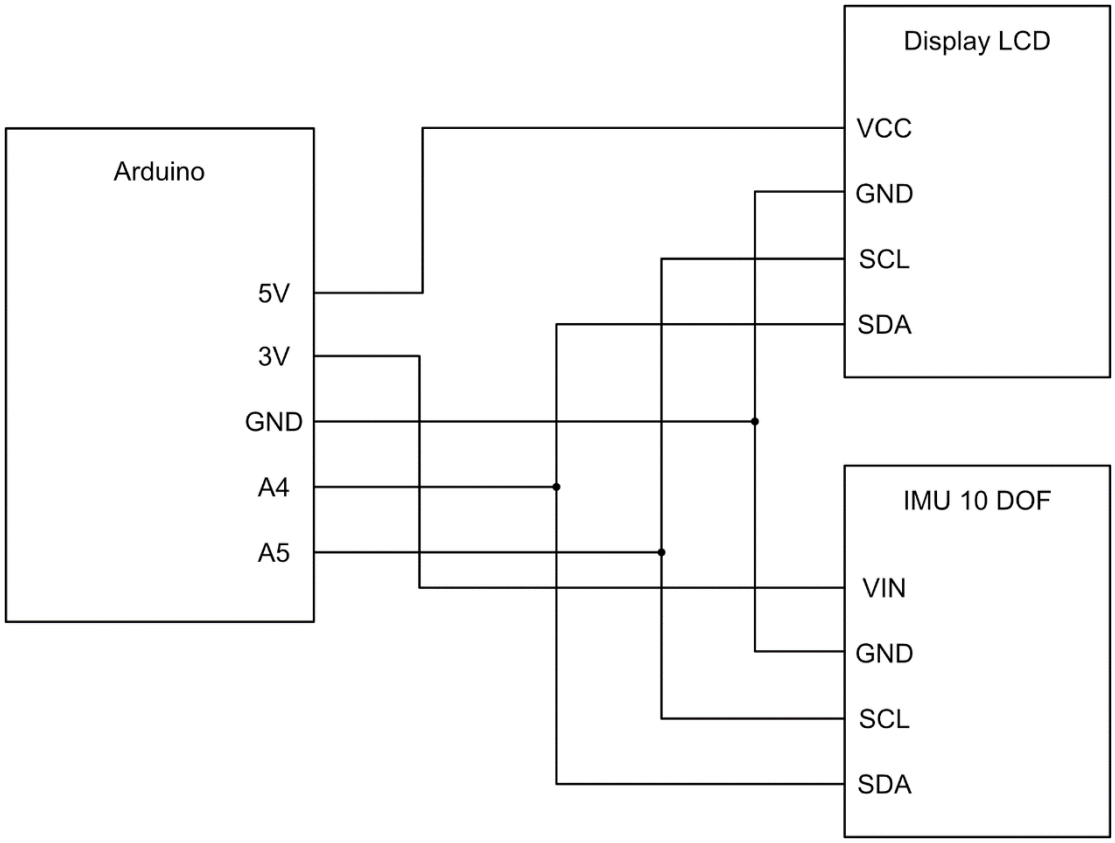

Figura 1 - Diagrama de Ligações

Como já referido, o objetivo é obter as medições da temperatura, pressão atmosférica e altitude, a cada dez segundos, e apresentar essa mesma informação já atualizada. Podemos recorrer assim a um diagrama de atividades onde teremos duas atividades: Leitura e Apresentação de Valores, que será uma só atividade, e a Esperar 10 Segundos.

A atividade Leitura e Apresentação de Valores é uma macro-atividade, ou seja, pode ser descrita por um diagrama de atividades mais elementar[1][2]. Este também possui duas atividades, Leitura de Valores e Escrita de Valores.

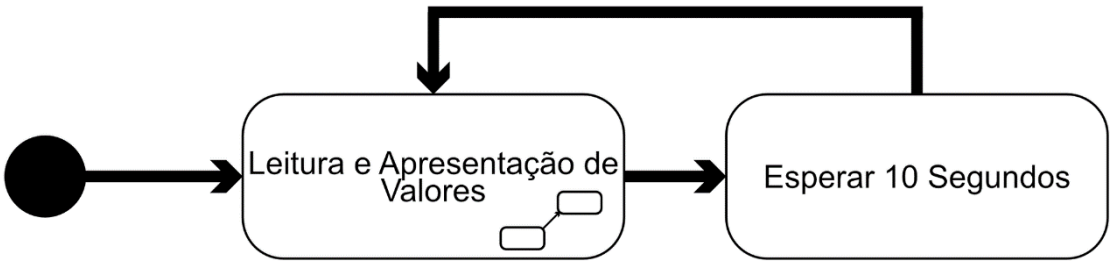

Figura 2 - Diagrama de Actividades

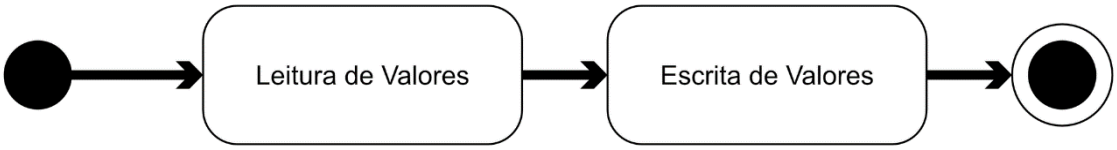

Figura 3 - Macro Diagrama de Actividades de Leitura e Apresentação de Valores

3. Protocolo I^2C

O protocolo I^2C é partilhado pela placa e pelo display. Este protocolo o envio de comandos e dados, assim como a leitura de dados[1][2].

No caso do display, enviamos comandos e dados enquanto a placa também devolve dados.

Para estas operações serem efetuadas, começamos por comunicar com um determinado endereço. Este é o endereço do componente em questão. De seguida enviamos ou um comando ou dados, que eventualmente serão processados como desejados. Para receber dados da placa, o mesmo processo é realizado, seguindo da leitura do que é recebido, caso realmente haja essa informação.

Usaremos a biblioteca Wire.h para facilitar a manipulação dos sinais SDA e SCL[1][2].

4. Comunicação Série

De forma a termos a simulação do outdoor, é necessária a comunicação série entre o Arduino e uma aplicação implementada em Python, com recurso ao uso da biblioteca Serial[1][2].

Desta forma, quando os valores da temperatura, pressão atmosférica e altitude são actualizados, é também enviado por comunicação série os mesmos valores. Estes serão lidos pela aplicação Python e depois apresentados na consola.

5. Montagem

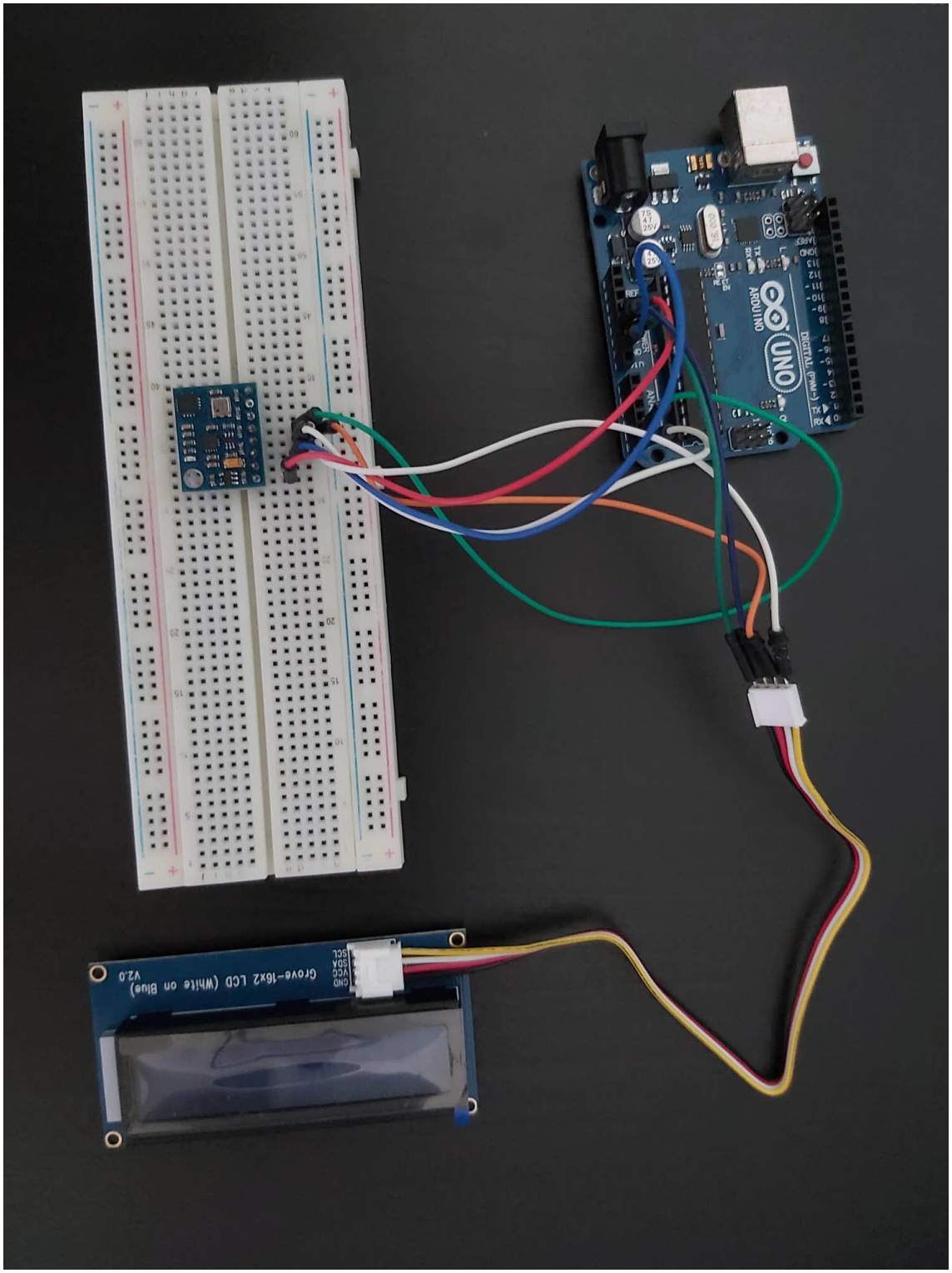

Figura 4 - Montagem Geral

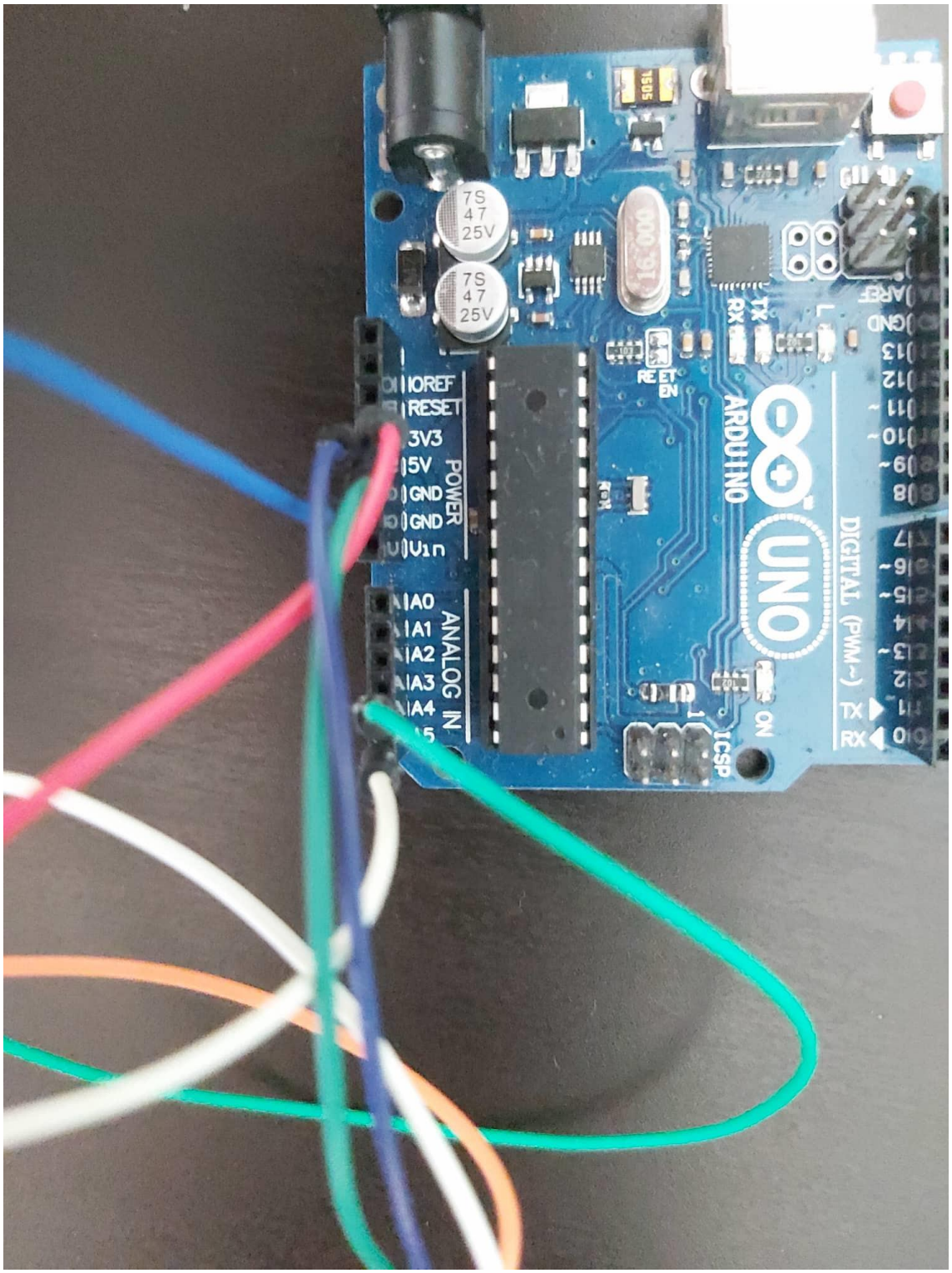

Figura 5 - Montagem Arduino

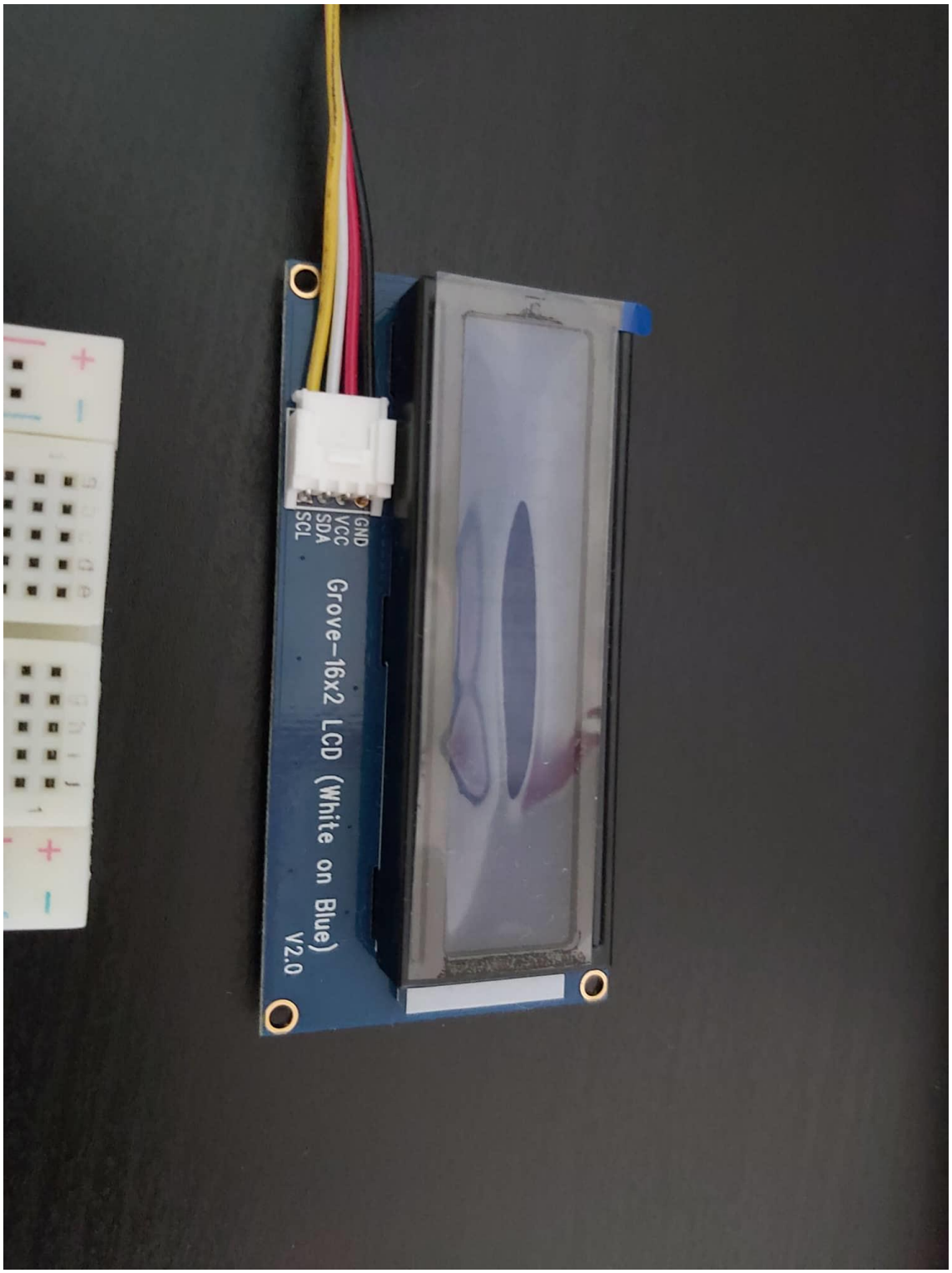

Figura 6 - Montagem Display

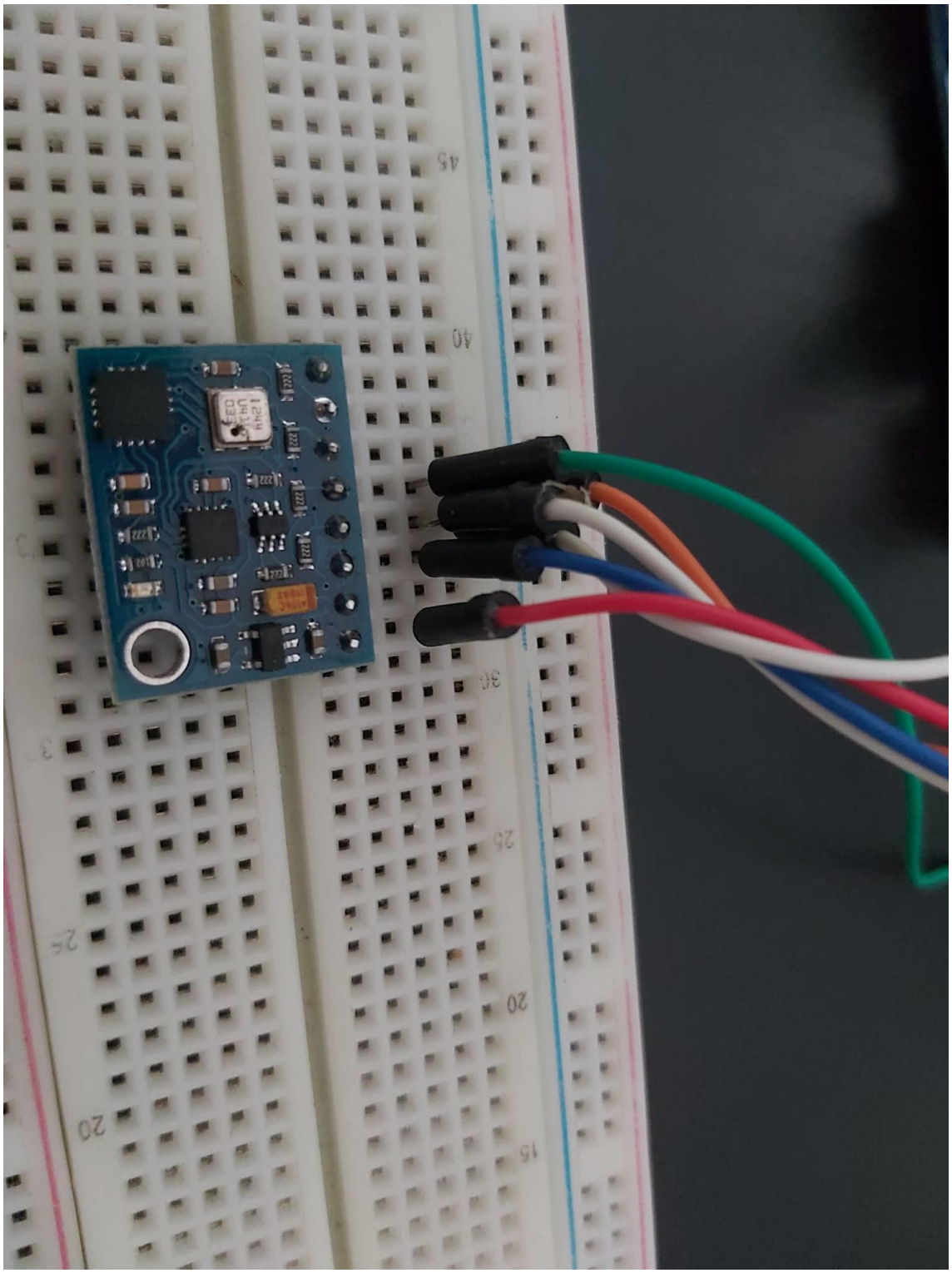

Figura 7 - Montagem Placa IMU 10 DOF

6. Resultados

Figura 8 - Resultado Display

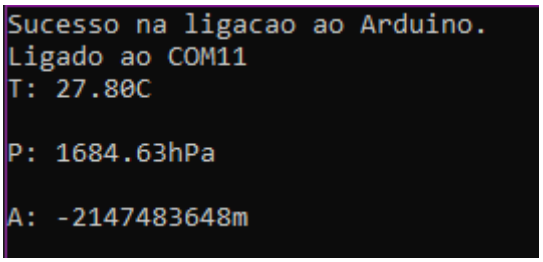

Figura 9 - Resultado Consola em Python

7. Implementação

7.1. Arduino

Como foi mencionado, é usado a biblioteca Wire.h.

```
#include <Wire.h>
```

Figura 10 - Biblioteca Wire.h

De seguida, definimos os endereços, tanto do display como do sensor BMP180, e ainda os endereços para o envio de comandos e de dados do display.

```
#define ENDERECO_DISPLAY 0x3E
#define COMANDOS 0x80
#define DADOS 0x40
#define ENDERECO_BMP180 0x77
```

Figura 11 - Definição de Endereços

Também temos de definir as variáveis globais, tanto para calcular os valores de temperatura, pressão atmosférica e altitude, assim como o estado do autómato. Temos ainda as variáveis para guardar os tempos de espera e atualizar os valores acima referidos.

```
short AC_1, AC_2, AC_3, B_1, B_2, MB, MC, MD;
unsigned short AC_4, AC_5, AC_6;
long UP, UT,T;
float P;
long H;
byte oss = 3;
byte delayOss = {5, 8, 14, 26};

boolean estado;
unsigned long now, ago;
```

Figura 12 - Variáveis Globais

Começamos a definir os métodos que serviram para enviar comandos e dados para o display. Como referido, definimos a comunicação com o endereço do display, enviamos o endereço para comandos ou para dados, e de seguida a operação que queremos executar, ou dados a apresentar.

```
//----- Métodos Display -----
void escreverComando8(byte oitoBits)
{
  Wire.beginTransmission(ENDERECO_DISPLAY);
  Wire.write(COMANDOS);
  Wire.write(oitoBits);
  Wire.endTransmission();
}

void escreverDados8(byte oitoBits)
{
  Wire.beginTransmission(ENDERECO_DISPLAY);
  Wire.write(DADOS);
  Wire.write(oitoBits);
  Wire.endTransmission();
}
```

Figura 13 - Métodos do Display

Temos de definir cada comando que o display executa. Neste caso, temos o *displayInit* que inicializa o display; o *displayClear* que apaga tudo o que o display está a exibir; *displaySetCursor* que define onde queremos colocar o cursor; *displayCursorOffBlinkOff* para esconder o cursor e para este não piscar; *displayCursorOnBlinkOn* para mostrar o cursor e fazê-lo piscar; *displayCursorHome* que insere o cursor na primeira coluna da primeira linha.

```
//Comandos
void displayInit()
{
  delay(30);
  escreverComando8(0x38);
  delayMicroseconds(50);
  escreverComando8(0x0F);
  delayMicroseconds(50);
  escreverComando8(0x01);
  delay(10);
  escreverComando8(0x06);
}

void displayClear()
{
  escreverComando8(0x1);
  delay(10);
}

void displaySetCursor(byte linha, byte coluna)
{
  escreverComando8(0b10000000 | linha << 6 | coluna);
  delayMicroseconds(50);
}
void displayCursorOffBlinkOff()
{
  escreverComando8(0x0C);
  delayMicroseconds(50);
}

void displayCursorOnBlinkOn()
{
  escreverComando8(0x0F);
  delayMicroseconds(50);
}

void displayCursorHome()
{
  escreverComando8(0x02);
  delay(10);
}
```

Figura 14 - Comandos do Display

Tal como para os comandos, também é necessário criar métodos que enviem dados para o display. Neste caso, o *displayPrintChar* que exibe um caracter no display; o *displayPrintString* que apresenta uma *string* no display.

```
//Dados
void displayPrintChar(char c)
{
  escreverDados8(c);
  delayMicroseconds(50);
}

void displayPrintString(String s)
{
  for(int charIndex = 0; charIndex < s.length(); charIndex++)
  {
    if(s.indexOf(charIndex) == '\0')
    {
      return;
    }

    displayPrintChar(s.charAt(charIndex));
  }
}
```

Figura 15 - Dados do Display

Tal como fizemos para o display, agora temos de repetir o processo para o sensor BMP180. Começamos por definir os métodos para a leitura e envio de dados.

```
// ----- Métodos BMP180 -----
short BMP180_read_16bit_value(byte regAdress)
{
  Wire.beginTransmission(ENDERECO_BMP180);
  Wire.write(regAdress);
  Wire.endTransmission();

  Wire.requestFrom(ENDERECO_BMP180, 2);
  while(Wire.available() == 0);
  byte MSB = Wire.read();
  while(Wire.available() == 0);
  byte LSB = Wire.read();
  Wire.endTransmission();

  return (word) MSB << 8 | LSB;
}

long BMP180_read_24bit_value(byte regAdress)
{
  Wire.beginTransmission(ENDERECO_BMP180);
  Wire.write(regAdress);
  Wire.endTransmission();

  Wire.requestFrom(ENDERECO_BMP180, 3);
  while(Wire.available() == 0);
  byte MSB = Wire.read();
  while(Wire.available() == 0);
  byte LSB = Wire.read();
  while(Wire.available() == 0);
  byte XLSB = Wire.read();
  Wire.endTransmission();

  return (long) MSB << 16 | (word) LSB << 8 | XLSB;
}

void BMP180_write_8bit_value(byte regAdress, byte value)
{
  Wire.beginTransmission(ENDERECO_BMP180);
  Wire.write(regAdress);
  Wire.write(value);
  Wire.endTransmission();
}
```

Figura 16 - Métodos do Sensor BMP180

Passamos para a implementação da leitura dos valores do sensor. O primeiro método é a leitura dos valores de calibração do sensor.

```
void BMP180_readCalibrationData()
{
  AC_1 = BMP180_read_16bit_value(0xAA);
  AC_2 = BMP180_read_16bit_value(0xAC);
  AC_3 = BMP180_read_16bit_value(0xAE);
  AC_4 = BMP180_read_16bit_value(0xB0);
  AC_5 = BMP180_read_16bit_value(0xB2);
  AC_6 = BMP180_read_16bit_value(0xB4);
  B_1 = BMP180_read_16bit_value(0xB6);
  B_2 = BMP180_read_16bit_value(0xB8);
  MB = BMP180_read_16bit_value(0xBA);
  MC = BMP180_read_16bit_value(0xBC);
  MD = BMP180_read_16bit_value(0xBE);
}
```

Figura 17 - Read Calibration Data

De seguida, obtemos o valor de temperatura antes de ser compensada.

```
short BMP180_readUncompensatedTemperatureValue()
{
  BMP180_write_8bit_value(0xF4, 0x2E);
  delay(4.5);
  return BMP180_read_16bit_value(0xF6);
}
```

Figura 18 - Read Uncompensated Temperature Value

É feito o mesmo para o valor da pressão atmosférica.

```
long BMP180_readUncompensatedPressureValue()
{
  BMP180_write_8bit_value(0xF4, (0x34 + (oss << 6)));
  delay(delayOss[oss]);
  return BMP180_read_24bit_value(0xF6) >> (8 - oss);
}
```

Figura 19 - Read Uncompensated Pressure Value

Após obtermos estes valores, vamos fazer as operações necessárias para obtermos os valores corretos da temperatura e pressão atmosférica.

```
void BMP180_calculateTrueTemperatureAndPressure()
{
  long X_1 = (UT - AC_6) * AC_5 / pow(2, 15);
  long X_2 = MC * pow(2, 11) / (X_1 + MD);
  long B_5 = X_1 + X_2;
  T = (B_5 + 8) / pow(2, 4);

  long B_6 = B_5 - 4000;
  X_1 = (B_2 * (B_6 * B_6 / pow(2, 12))) / pow(2, 11);
  X_2 = AC_2 * B_6 / pow(2, 11);
  long X_3 = X_1 + X_2;
  long B_3 = (((AC_1 * 4 + X_3) << oss) + 2) / 4;
  X_1 = AC_3 + B_6 / pow(2, 13);
  X_2 = (B_1 * (B_6 * B_6 / pow(2, 12))) / pow(2, 16);
  X_3 = ((X_1 + X_2) + 2) / pow(2, 2);
  unsigned long B_4 = AC_4 * (unsigned long)(X_3 + 32768) / pow(2, 15);
  unsigned long B_7 = ((unsigned long) UP - B_3) * (50000 >> oss);
  if(B_7 < 0x80000000)
  {
    P = (B_7 * 2) / B_4;
  }
  else
  {
    P = (B_7 / B_4) * 2;
  }
  X_1 = (P / pow(2, 8)) * (P / pow(2, 8));
  X_1 = (X_1 * 3038) / pow(2, 16);
  X_2 = (-7357 * P) / pow(2, 16);
  P = P + (X_1 + X_2 + 3791) / pow(2, 4);
  P = P / 100.;
}
```

Figura 20 - Calculate True Temperature And Pressure

Por fim, obtemos o valor da altitude, a partir do valor da pressão atmosférica.

```
float BMP180_calculateAltitude()
{
  return 44330 * (1 - pow((P / 1013.25),1/5.255));
}
```

Figura 21 - Calculate Altitude

Como foi apresentado previamente, o objectivo é implementar um autómato onde obtém os valores actuais da temperatura, pressão atmosférica e altitude, apresenta estes valores e depois entra num estado suspenso durante dez segundos, repetindo depois o processo.

Para facilitar este objectivo, foram criados dois métodos, um que lê e exibe os valores, e outro que envia, via comunicação série, os mesmos valores.

```
void displayCommunication()
{
  UT = BMP180_readUncompensatedTemperatureValue();
  UP = BMP180_readUncompensatedPressureValue();
  BMP180_calculateTrueTemperatureAndPressure();
  H = BMP180_calculateAltitude();

  displayClear();
  displayPrintString("T: ");
  displayPrintString(String((float)T/10.));
  displayPrintChar(0xDF); // Char: °
  displayPrintChar('C');
  displaySetCursor(1, 0);
  displayPrintString("P: ");
  displayPrintString(String(P));
  displayPrintString("hPa");
}
```

Figura 22 - Display Communication

```
void serialCommunication()
{
  Serial.print("T: ");
  Serial.print(T/10.);
  Serial.println("°C");

  Serial.print("P: ");
  Serial.print(P);
  Serial.println("hPa");

  Serial.print("A: ");
  Serial.print(H);
  Serial.println("m");
}
```

Figura 23 - Serial Communication

No nosso *setup*, iniciamos a comunicação série. O *baud rate* tem de ser o mesmo que na aplicação feita em Python. Inicializamos o arduino como o mestre no protocolo I^2C. Inicializamos também o display e escondemos o cursor. Lemos os valores de calibração do sensor BMP180 e definimos o estado do autómato para verdadeiro, pois só vamos ter dois estados que podem ser diferenciados pelo valor booleano.

```
void setup()
{
  Serial.begin(9600);
  Wire.begin();

  //Display
  displayInit();
  displayCursorOffBlinkOff();

  //BMP180
  BMP180_readCalibrationData();

  estado = true;
}
```

Figura 24 - Setup

No *loop* ocorre o autómato. Este verifica o estado actual, se for o correspondente à atividade Esperar, calculamos o tempo que passou desde a última atualização. Se esse intervalo de tempo for igual ou superior a dez segundos, o autómato evolui no sentido da atividade Leitura e Apresentação de Valores. Estes valores são atualizados, exibidos e enviados via comunicação série e atualizado o momento em que saímos desse estado. No final de cada uma destas atividades, o estado é atualizado para o complementado.

```
void loop()
{
  switch(estado)
  {
    case false:

      now = millis();

      if(now - ago >= 10000)
      {
        estado = true;
      }

      break;

     case true:

      displayCommunication();
      serialCommunication();

      ago = millis();
      estado = false;
      break;
  }
}
```

Figura 25 - Loop

7.2. Python

Como já referido anteriormente, para usarmos a comunicação série, temos de importar a biblioteca *serial* do Python.

```
import serial
```

Figura 26 - Importar Biblioteca Serial

Criamos duas variáveis que igualam a porta onde o Arduino está ligado e, como já mencionado, o *baud rate*.

```
com = 'COM11' # Tem de coincidir com o do Arduino
baudrate = 9600 # Tem de coincidir com o do Arduino
```

Figura 27 – Variáveis

De seguida, implementamos o método que inicializa a comunicação série com o Arduino.

```
# Função de inicialização
def comInit(com, baudrate):
    try:
        Serie = serial.Serial(com, baudrate)
        print('Sucesso na ligacao ao Arduino.')
        print ('Ligado ao ' + Serie.portstr)
        return Serie
    except Exception as e:
        print ('Insucesso na ligacao ao Arduino.')
        print (e)
        return None
```

Figura 28 - Método de Inicialização da Comunicação Série

Definimos também o método que recebe informação pela comunicação Série.

```
def stringReceive(Serie):
    try:
        return Serie.readline().strip()
    except Exception as e:
        print ('Erro na comunicacao (stringReceive).')
        print (e)
        Serie.close()
```

Figura 29 - Método de Recepção de Informação

Finalmente, temos o *loop* do programa que verifica se existe alguma informação a receber e, caso exista, exibe-a na consola da aplicação.

```
#Programa principal
s = comInit(com, baudrate)
while(True):
    print(stringReceive(s))
    print()
```

Figura 30 - Loop do Programa Python

8. Código

8.1. Arduino

```
#include <Wire.h>

#define ENDERECO_DISPLAY 0x3E
#define COMANDOS 0x80
#define DADOS 0x40
#define ENDERECO_BMP180 0x77

short AC_1, AC_2, AC_3, B_1, B_2, MB, MC, MD;
unsigned short AC_4, AC_5, AC_6;
long UP, UT,T;
float P;
long H;
byte oss = 3;
byte delayOss = {5, 8, 14, 26};

boolean estado;
unsigned long now, ago;

//----- Métodos Display -----
void escreverComando8(byte oitoBits)
{
  Wire.beginTransmission(ENDERECO_DISPLAY);
  Wire.write(COMANDOS);
  Wire.write(oitoBits);
  Wire.endTransmission();
}

void escreverDados8(byte oitoBits)
{
  Wire.beginTransmission(ENDERECO_DISPLAY);
  Wire.write(DADOS);
  Wire.write(oitoBits);
  Wire.endTransmission();
}

//Comandos
void displayInit()
{
  delay(30);
  escreverComando8(0x38);
  delayMicroseconds(50);
  escreverComando8(0x0F);
  delayMicroseconds(50);
  escreverComando8(0x01);
  delay(10);
```

```
  escreverComando8(0x06);
}

void displayClear()
{
  escreverComando8(0x1);
  delay(10);
}

void displaySetCursor(byte linha, byte coluna)
{
  escreverComando8(0b10000000 | linha << 6 | coluna);
  delayMicroseconds(50);
}

void displayCursorOffBlinkOff()
{
  escreverComando8(0x0C);
  delayMicroseconds(50);
}

void displayCursorOnBlinkOn()
{
  escreverComando8(0x0F);
  delayMicroseconds(50);
}

void displayCursorHome()
{
  escreverComando8(0x02);
  delay(10);
}

//Dados
void displayPrintChar(char c)
{
  escreverDados8(c);
  delayMicroseconds(50);
}
```

```
void displayPrintString(String s)
{
  for(int charIndex = 0; charIndex < s.length(); charIndex++)
  {
    if(s.indexOf(charIndex) == '\0')
    {
      return;
    }

    displayPrintChar(s.charAt(charIndex));
  }
}
// ----- Métodos Display -----

// ----- Métodos BMP180 -----
short BMP180_read_16bit_value(byte regAdress)
{
  Wire.beginTransmission(ENDERECO_BMP180);
  Wire.write(regAdress);
  Wire.endTransmission();

  Wire.requestFrom(ENDERECO_BMP180, 2);
  while(Wire.available() == 0);
  byte MSB = Wire.read();
  while(Wire.available() == 0);
  byte LSB = Wire.read();
  Wire.endTransmission();

  return (word) MSB << 8 | LSB;
}

long BMP180_read_24bit_value(byte regAdress)
{
  Wire.beginTransmission(ENDERECO_BMP180);
  Wire.write(regAdress);
  Wire.endTransmission();

  Wire.requestFrom(ENDERECO_BMP180, 3);
  while(Wire.available() == 0);
  byte MSB = Wire.read();
  while(Wire.available() == 0);
  byte LSB = Wire.read();
  while(Wire.available() == 0);
  byte XLSB = Wire.read();
  Wire.endTransmission();

  return (long) MSB << 16 | (word) LSB << 8 | XLSB;
```

```
}

void BMP180_write_8bit_value(byte regAdress, byte value)
{
  Wire.beginTransmission(ENDERECO_BMP180);
  Wire.write(regAdress);
  Wire.write(value);
  Wire.endTransmission();
}

void BMP180_readCalibrationData()
{
  AC_1 = BMP180_read_16bit_value(0xAA);
  AC_2 = BMP180_read_16bit_value(0xAC);
  AC_3 = BMP180_read_16bit_value(0xAE);
  AC_4 = BMP180_read_16bit_value(0xB0);
  AC_5 = BMP180_read_16bit_value(0xB2);
  AC_6 = BMP180_read_16bit_value(0xB4);
  B_1 = BMP180_read_16bit_value(0xB6);
  B_2 = BMP180_read_16bit_value(0xB8);
  MB = BMP180_read_16bit_value(0xBA);
  MC = BMP180_read_16bit_value(0xBC);
  MD = BMP180_read_16bit_value(0xBE);
}

short BMP180_readUncompensatedTemperatureValue()
{
  BMP180_write_8bit_value(0xF4, 0x2E);
  delay(5);
  return BMP180_read_16bit_value(0xF6);
}

long BMP180_readUncompensatedPressureValue()
{
  BMP180_write_8bit_value(0xF4, (0x34 + (oss << 6)));
  delay(delayOss[oss]);
  return BMP180_read_24bit_value(0xF6) >> (8 - oss);
}
```

```
void BMP180_calculateTrueTemperatureAndPressure()
{
  long X_1 = (UT - AC_6) * AC_5 / pow(2, 15);
  long X_2 = MC * pow(2, 11) / (X_1 + MD);
  long B_5 = X_1 + X_2;
  T = (B_5 + 8) / pow(2, 4);

  long B_6 = B_5 - 4000;
  X_1 = (B_2 * (B_6 * B_6 / pow(2, 12))) / pow(2, 11);
  X_2 = AC_2 * B_6 / pow(2, 11);
  long X_3 = X_1 + X_2;
  long B_3 = (((AC_1 * 4 + X_3) << oss) + 2) / 4;
  X_1 = AC_3 + B_6 / pow(2, 13);
  X_2 = (B_1 * (B_6 * B_6 / pow(2, 12))) / pow(2, 16);
  X_3 = ((X_1 + X_2) + 2) / pow(2, 2);
  unsigned long B_4 = AC_4 * (unsigned long)(X_3 + 32768) /
pow(2, 15);
  unsigned long B_7 = ((unsigned long) UP - B_3) * (50000 >>
oss);
  if(B_7 < 0x80000000)
  {
    P = (B_7 * 2) / B_4;
  }
  else
  {
    P = (B_7 / B_4) * 2;
  }
  X_1 = (P / pow(2, 8)) * (P / pow(2, 8));
  X_1 = (X_1 * 3038) / pow(2, 16);
  X_2 = (-7357 * P) / pow(2, 16);
  P = P + (X_1 + X_2 + 3791) / pow(2, 4);
  P = P / 100.;
}

float BMP180_calculateAltitude()
{
  return 44330 * (1 - pow((P / 1013.25),1/5.255));
}

// ----- Métodos BMP180 -----
```

```
void setup()
{
  Serial.begin(9600);
  Wire.begin();

  //Display
  displayInit();
  displayCursorOffBlinkOff();

  //BMP180
  BMP180_readCalibrationData();

  estado = true;
}

void loop()
{
  switch(estado)
  {
    case false:

      now = millis();

      if(now - ago >= 10000)
      {
        estado = true;
      }

      break;

     case true:

      displayCommunication();
      serialCommunication();

      ago = millis();
      estado = false;
      break;
  }
}
```

```
void serialCommunication()
{
  Serial.print("T: ");
  Serial.print(T/10.);
  Serial.println("ºC");

  Serial.print("P: ");
  Serial.print(P);
  Serial.println("hPa");

  Serial.print("A: ");
  Serial.print(H);
  Serial.println("m");
}

void displayCommunication()
{
  UT = BMP180_readUncompensatedTemperatureValue();
  UP = BMP180_readUncompensatedPressureValue();
  BMP180_calculateTrueTemperatureAndPressure();
  H = BMP180_calculateAltitude();

  displayClear();
  displayPrintString("T:");
  displayPrintString(String((float)T/10.));
  displayPrintChar(0xDF); // Char: º
  displayPrintChar('C');
  displayPrintString(" A:");
  displayPrintString(String(H));
  displayPrintChar('m');
  displaySetCursor(1, 0);
  displayPrintString("P:");
  displayPrintString(String(P));
  displayPrintString("hPa");
}
```

8.2. Python

```
# -*- coding: cp1252 -*-
import serial

com = 'COM11' # Tem de coincidir com o do Arduino
baudrate = 9600 # Tem de coincidir com o do Arduino

# Função de inicialização
def comInit(com, baudrate):
    try:
        Serie = serial.Serial(com, baudrate)
        print('Sucesso na ligacao ao Arduino.')
        print ('Ligado ao ' + Serie.portstr)
        return Serie
    except Exception as e:
        print ('Insucesso na ligacao ao Arduino.')
        print (e)
        return None

def stringReceive(Serie):
    try:
        return Serie.readline().strip()
    except Exception as e:
        print ('Erro na comunicacao (stringReceive).')
        print (e)
        Serie.close()

#Programa principal
s = comInit(com, baudrate)
while(True):
    print(stringReceive(s))
    print()
```

9. Observações

Após algumas observações, foi possível concluir que a placa IMU 10 DOF apresenta uma avaria pois os valores devolvidos da pressão atmosférica são extremamente altos, o que implicaria que estaríamos vários metros debaixo do nível do mar.

Dado isto, na aula foi combinada uma pequena solução que exibia os resultados corretos. Esta foi feita por uma pequena modificação no código, o que serviu para os valores ficarem corretos.

```
P -= 670;
```

Figura 31 - Alteração no método BMP180_calculateTrueTemperatureAndPressure

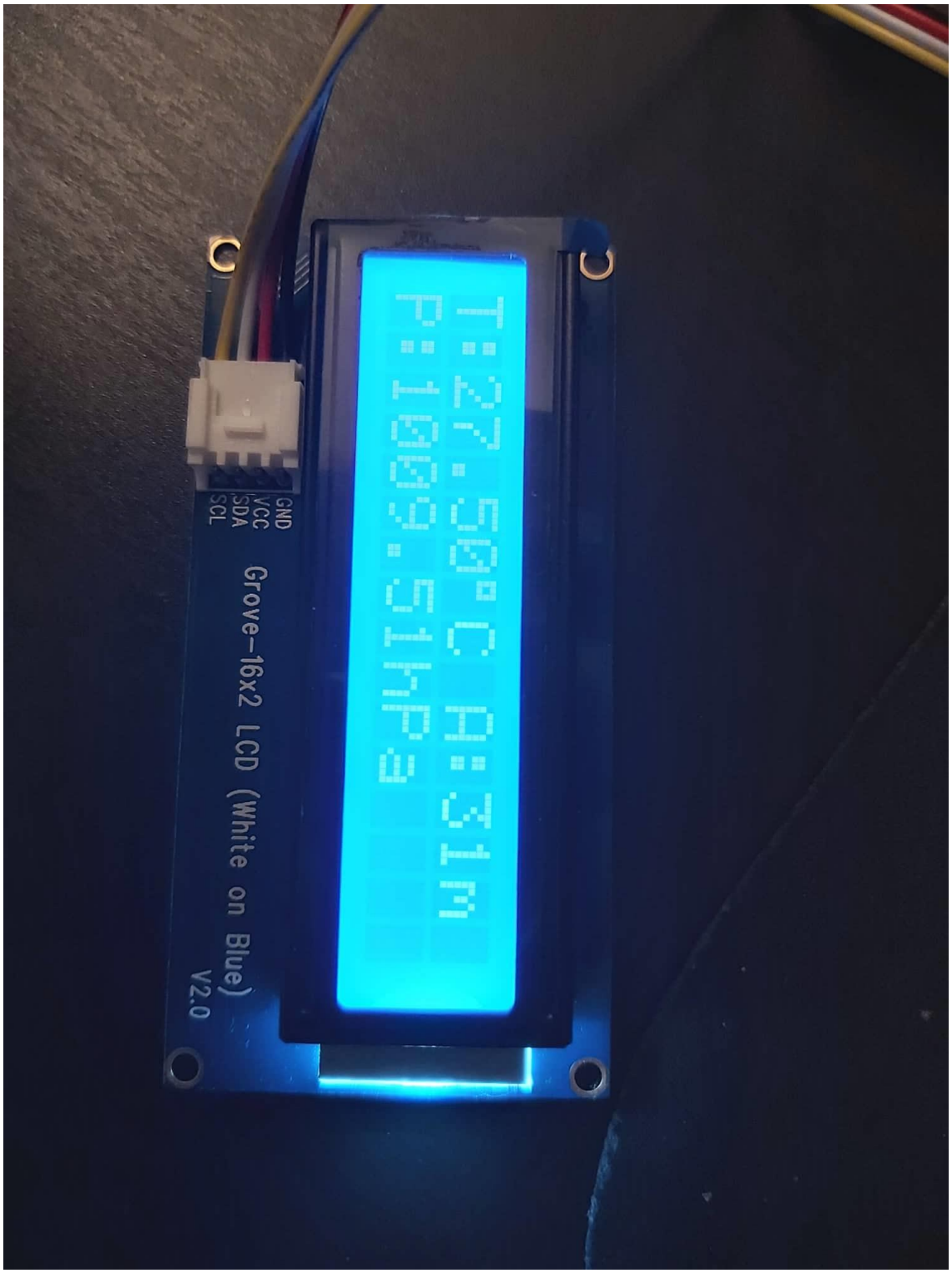

Figura 32 - Display Valores Corretos

10. Conclusões

Com este trabalho finalizado, foi possível aprofundar os conhecimentos relacionados com o protocolo I^2C, assim como a comunicação série. Foi também possível experienciar como os display's funcionam no nosso quotidiano assim como os *outdoors*.

11. Bibliografia

[1] Carlos Carvalho (2022). *Computação Física*.

[2] Jorge Pais (2022). *Computação Física*.